OpenShift Dedicated 4

Monitoring user-defined projects

Monitoring user-defined projects in OpenShift Dedicated

Last Updated: 2021-10-27

OpenShift Dedicated 4 Monitoring user-defined projects

Monitoring user-defined projects in OpenShift Dedicated

Legal Notice

Abstract

This document provides instructions about using the optional feature to monitor pods and services in user-defined projects in OpenShift Dedicated clusters.

Table of Contents

CHAPTER 1. UNDERSTANDING THE MONITORING STACK

In OpenShift Dedicated, you can monitor your own projects in isolation from Red Hat Site Reliability Engineer (SRE) platform metrics. You can monitor your own projects without the need for an additional monitoring solution.

NOTE

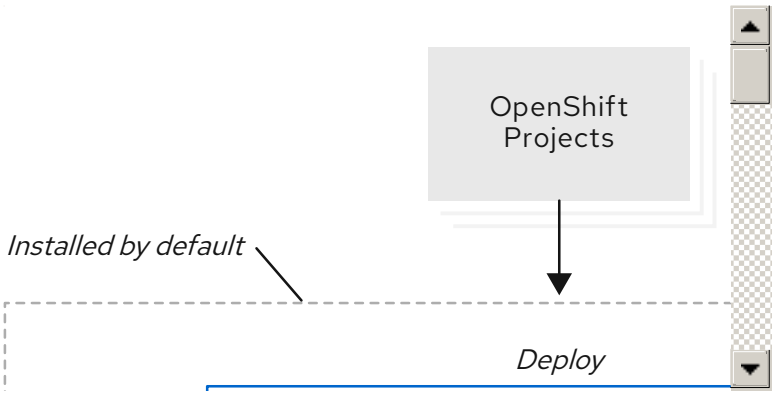
Follow the instructions in this document carefully to configure a supported Prometheus instance for monitoring user-defined projects. Custom Prometheus instances are not supported by OpenShift Dedicated.

1.1. UNDERSTANDING THE MONITORING STACK

The OpenShift Dedicated monitoring stack is based on the Prometheus open source project and its wider ecosystem. The monitoring stack includes the following:

- **Default platform monitoring components**. A set of platform monitoring components are installed in the **openshift-monitoring** project by default during an OpenShift Dedicated installation. This provides monitoring for core OpenShift Dedicated. The default monitoring stack also enables remote health monitoring for clusters. Critical metrics, such as CPU and memory, are collected from all of the workloads in every namespace and are made available for your use.
  These components are illustrated in the **Installed by default** section in the following diagram.

- **Components for monitoring user-defined projects** This feature is enabled by default and provides monitoring for user-defined projects. These components are illustrated in the **User** section in the following diagram.



1.1.1. Components for monitoring user-defined projects

OpenShift Dedicated includes an optional enhancement to the monitoring stack that enables you to monitor services and pods in user-defined projects. This feature includes the following components:

Table 1.1. Components for monitoring user-defined projects

Component	Description
Prometheus Operator	The Prometheus Operator in the **openshift-user-workload-monitoring** project creates, configures, and manages Prometheus and Thanos Ruler instances in the same project.

Component	Description
Prometheus	Prometheus is the monitoring system through which monitoring is provided for user-defined projects. Prometheus sends alerts to Alertmanager for processing. However, alert routing is not currently supported.
Thanos Ruler	The Thanos Ruler is a rule evaluation engine for Prometheus that is deployed as a separate process. In OpenShift Dedicated 4, Thanos Ruler provides rule and alerting evaluation for the monitoring of user-defined projects.

All of these components are monitored by the stack and are automatically updated when OpenShift Dedicated is updated.

1.1.2. Monitoring targets for user-defined projects

Monitoring is enabled by default for OpenShift Dedicated user-defined projects. You can monitor:

- Metrics provided through service endpoints in user-defined projects.
- Pods running in user-defined projects.

1.2. ADDITIONAL RESOURCES

- Accessing monitoring for user-defined projects
- Default monitoring components
- Default monitoring targets

1.3. NEXT STEPS

- Configuring the monitoring stack

CHAPTER 2. CONFIGURING THE MONITORING STACK

This document explains what is supported for the monitoring of user-defined projects. It also shows how to configure the monitoring stack, and demonstrates several common configuration scenarios.

2.1. MAINTENANCE AND SUPPORT FOR MONITORING

The supported way of configuring OpenShift Dedicated Monitoring is by using the options described in this document. **Do not use other configurations, as they are unsupported.**

IMPORTANT

Installing another Prometheus instance is not supported by the Red Hat Site Reliability Engineers (SREs).

Configuration paradigms can change across Prometheus releases, and such cases can only be handled gracefully if all configuration possibilities are controlled. If you use configurations other than those described in this section, your changes will disappear because the **cluster-monitoring-operator** reconciles any differences. The Operator resets everything to the defined state by default and by design.

2.1.1. Support considerations for monitoring user-defined projects

The following modifications are explicitly not supported:

- Installing custom Prometheus instances on OpenShift Dedicated

2.2. CONFIGURING THE MONITORING STACK

In OpenShift Dedicated, you can configure the stack that monitors workloads for user-defined projects by using the **user-workload-monitoring-config ConfigMap** object. Config maps configure the Cluster Monitoring Operator (CMO), which in turn configures the components of the stack.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have created the **user-workload-monitoring-config ConfigMap** object.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **ConfigMap** object.
   a. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

      ```
      $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
      ```

   b. Add your configuration under **data.config.yaml** as a key-value pair **<component_name>: <component_configuration>**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>:
      <configuration_for_the_component>
```

Substitute **<component>** and **<configuration_for_the_component>** accordingly.

The following example **ConfigMap** object configures a data retention period and minimum container resource requests for Prometheus. This relates to the Prometheus instance that monitors user-defined projects only:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus: 1
      retention: 24h 2
      resources:
        requests:
          cpu: 200m 3
          memory: 2Gi 4
```

1. Defines the Prometheus component and the subsequent lines define its configuration.
2. Configures a 24 hour data retention period for the Prometheus instance that monitors user-defined projects.
3. Defines a minimum resource request of 200 millicores for the Prometheus container.
4. Defines a minimum pod resource request of 2 GiB of memory for the Prometheus container.

2. Save the file to apply the changes to the **ConfigMap** object. The pods affected by the new configuration are restarted automatically.

WARNING

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

2.3. CONFIGURABLE MONITORING COMPONENTS

This table shows the monitoring components you can configure and the keys used to specify the components in the **user-workload-monitoring-config ConfigMap** objects:

Table 2.1. Configurable monitoring components

Component	user-workload-monitoring-config config map key
Prometheus Operator	**prometheusOperator**
Prometheus	**prometheus**
Thanos Ruler	**thanosRuler**

2.4. MOVING MONITORING COMPONENTS TO DIFFERENT NODES

You can move any of the components that monitor workloads for user-defined projects to specific worker nodes. It is not permitted to move components to master or infrastructure nodes.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have created the **user-workload-monitoring-config ConfigMap** object.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. To move a component that monitors user-defined projects, edit the **ConfigMap** object:
   a. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

   b. Specify the **nodeSelector** constraint for the component under **data.config.yaml**:

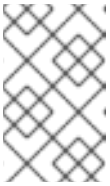
```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>:
      nodeSelector:
        <node_key>: <node_value>
        <node_key>: <node_value>
        <...>
```

Substitute **<component>** accordingly and substitute **<node_key>: <node_value>** with the map of key-value pairs that specifies the destination nodes. Often, only a single key-value pair is used.

The component can only run on nodes that have each of the specified key-value pairs as labels. The nodes can have additional labels as well.

IMPORTANT

Many of the monitoring components are deployed by using multiple pods across different nodes in the cluster to maintain high availability. When moving monitoring components to labeled nodes, ensure that enough matching nodes are available to maintain resilience for the component. If only one label is specified, ensure that enough nodes contain that label to distribute all of the pods for the component across separate nodes. Alternatively, you can specify multiple labels each relating to individual nodes.



NOTE

If monitoring components remain in a **Pending** state after configuring the **nodeSelector** constraint, check the pod logs for errors relating to taints and tolerations.

For example, to move monitoring components for user-defined projects to specific worker nodes labeled **nodename: worker1**, **nodename: worker2**, and **nodename: worker2**, use:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheusOperator:
      nodeSelector:
        nodename: worker1
    prometheus:
      nodeSelector:
        nodename: worker1
        nodename: worker2
    thanosRuler:
      nodeSelector:
        nodename: worker1
        nodename: worker2
```

2. Save the file to apply the changes. The components affected by the new configuration are moved to the new nodes automatically.

WARNING

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

Additional resources

- Understanding how to update labels on nodes
- Placing pods on specific nodes using node selectors
- See the Kubernetes documentation for details on the **nodeSelector** constraint

2.5. ASSIGNING TOLERATIONS TO COMPONENTS THAT MONITOR USER-DEFINED PROJECTS

You can assign tolerations to the components that monitor user-defined projects, to enable moving them to tainted worker nodes. Scheduling is not permitted on master or infrastructure nodes.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have created the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** namespace.
- The OpenShift CLI (**oc**) is installed.

Procedure

1. Edit the **ConfigMap** object:
   a. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

   b. Specify **tolerations** for the component:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
```

```
<component>:
  tolerations:
    <toleration_specification>
```

Substitute **<component>** and **<toleration_specification>** accordingly.

For example, **oc adm taint nodes node1 key1=value1:NoSchedule** adds a taint to **node1** with the key **key1** and the value **value1**. This prevents monitoring components from deploying pods on **node1** unless a toleration is configured for that taint. The following example configures the **thanosRuler** component to tolerate the example taint:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      tolerations:
      - key: "key1"
        operator: "Equal"
        value: "value1"
        effect: "NoSchedule"
```

2. Save the file to apply the changes. The new component placement configuration is applied automatically.

WARNING

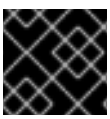
When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

Additional resources

- See the OpenShift Container Platform documentation on taints and tolerations
- See the Kubernetes documentation on taints and tolerations

2.6. CONFIGURING PERSISTENT STORAGE

Running cluster monitoring with persistent storage means that your metrics are stored to a persistent volume (PV) and can survive a pod being restarted or recreated. This is ideal if you require your metrics data to be guarded from data loss. For production environments, it is highly recommended to configure persistent storage. Because of the high IO demands, it is advantageous to use local storage.



IMPORTANT

See Recommended configurable storage technology .

2.6.1. Persistent storage prerequisites

- Use the block type of storage.

2.6.2. Configuring a local persistent volume claim

For monitoring components to use a persistent volume (PV), you must configure a persistent volume claim (PVC).

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have created the **user-workload-monitoring-config ConfigMap** object.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. To configure a PVC for a component that monitors user-defined projects, edit the **ConfigMap** object:

   a. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

   ```
   $ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
   ```

   b. Add your PVC configuration for the component under **data.config.yaml**:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: user-workload-monitoring-config
     namespace: openshift-user-workload-monitoring
   data:
     config.yaml: |
       <component>:
         volumeClaimTemplate:
           spec:
             storageClassName: <storage_class>
             resources:
               requests:
                 storage: <amount_of_storage>
   ```

   See the Kubernetes documentation on PersistentVolumeClaims for information on how to specify **volumeClaimTemplate**.

   The following example configures a PVC that claims local persistent storage for the Prometheus instance that monitors user-defined projects:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: user-workload-monitoring-config
   ```

```
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 40Gi
```

In the above example, the storage class created by the Local Storage Operator is called **local-storage**.

The following example configures a PVC that claims local persistent storage for Thanos Ruler:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    thanosRuler:
      volumeClaimTemplate:
        spec:
          storageClassName: local-storage
          resources:
            requests:
              storage: 40Gi
```

2. Save the file to apply the changes. The pods affected by the new configuration are restarted automatically and the new storage configuration is applied.

WARNING

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

2.6.3. Modifying the retention time for Prometheus metrics data

By default, the OpenShift Dedicated monitoring stack configures the retention time for Prometheus data to be 15 days. You can modify the retention time for the Prometheus instance that monitors user-defined projects, to change how soon the data is deleted.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.

- You have created the **user-workload-monitoring-config ConfigMap** object.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. To modify the retention time for the Prometheus instance that monitors user-defined projects, edit the **ConfigMap** object:
   a. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

   b. Add your retention time configuration under **data.config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      retention: <time_specification>
```

   Substitute **<time_specification>** with a number directly followed by **ms** (milliseconds), **s** (seconds), **m** (minutes), **h** (hours), **d** (days), **w** (weeks), or **y** (years).

   The following example sets the retention time to 24 hours for the Prometheus instance that monitors user-defined projects:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      retention: 24h
```

2. Save the file to apply the changes. The pods affected by the new configuration are restarted automatically.

WARNING

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

Additional resources

- Understanding persistent storage
- Optimizing storage

2.7. CONTROLLING THE IMPACT OF UNBOUND METRICS ATTRIBUTES IN USER-DEFINED PROJECTS

Developers can create labels to define attributes for metrics in the form of key-value pairs. The number of potential key-value pairs corresponds to the number of possible values for an attribute. An attribute that has an unlimited number of potential values is called an unbound attribute. For example, a **customer_id** attribute is unbound because it has an infinite number of possible values.

Every assigned key-value pair has a unique time series. The use of many unbound attributes in labels can result in an exponential increase in the number of time series created. This can impact Prometheus performance and can consume a lot of disk space.

A **dedicated-admin** can use the following measure to control the impact of unbound metrics attributes in user-defined projects:

- **Limit the number of samples that can be accepted**per target scrape in user-defined projects

NOTE

Limiting scrape samples can help prevent the issues caused by adding many unbound attributes to labels. Developers can also prevent the underlying cause by limiting the number of unbound attributes that they define for metrics. Using attributes that are bound to a limited set of possible values reduces the number of potential key-value pair combinations.

2.7.1. Setting a scrape sample limit for user-defined projects

You can limit the number of samples that can be accepted per target scrape in user-defined projects.

WARNING

If you set a sample limit, no further sample data is ingested for that target scrape after the limit is reached.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have created the **user-workload-monitoring-config ConfigMap** object.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

2. Add the **enforcedSampleLimit** configuration to **data.config.yaml** to limit the number of samples that can be accepted per target scrape in user-defined projects:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    prometheus:
      enforcedSampleLimit: 50000 1
```

1 A value is required if this parameter is specified. This **enforcedSampleLimit** example limits the number of samples that can be accepted per target scrape in user-defined projects to 50,000.

3. Save the file to apply the changes. The limit is applied automatically.

WARNING

When changes are saved to the **user-workload-monitoring-config ConfigMap** object, the pods and other resources in the **openshift-user-workload-monitoring** project might be redeployed. The running monitoring processes in that project might also be restarted.

Additional resources

- Determining why Prometheus is consuming a lot of disk space for steps to query which metrics have the highest number of scrape samples

2.8. SETTING LOG LEVELS FOR MONITORING COMPONENTS

You can configure the log level for Prometheus Operator, Prometheus, and Thanos Ruler.

The following log levels can be applied to each of those components in the **user-workload-monitoring-config ConfigMap** object:

- **debug**. Log debug, informational, warning, and error messages.
- **info**. Log informational, warning, and error messages.
- **warn**. Log warning and error messages only.
- **error**. Log error messages only.

The default log level is **info**.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role.
- You have created the **user-workload-monitoring-config ConfigMap** object.
- You have installed the OpenShift CLI (**oc**).

Procedure

1. Edit the **ConfigMap** object:
   a. Edit the **user-workload-monitoring-config ConfigMap** object in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring edit configmap user-workload-monitoring-config
```

   b. Add **logLevel: <log_level>** for a component under **data.config.yaml**:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-workload-monitoring-config
  namespace: openshift-user-workload-monitoring
data:
  config.yaml: |
    <component>: 1
      logLevel: <log_level> 2
```

   1 The monitoring component that you are applying a log level to.

   2 The log level to apply to the component.

2. Save the file to apply the changes. The pods for the component restarts automatically when you apply the log-level change.

WARNING

When changes are saved to a monitoring config map, the pods and other resources in the related project might be redeployed. The running monitoring processes in that project might also be restarted.

3. Confirm that the log level has been applied by reviewing the deployment or pod configuration in the related project. The following example checks the log level in the **prometheus-operator** deployment in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring get deploy prometheus-operator -o yaml |  grep "log-level"
```

Example output

```
        - --log-level=debug
```

4. Check that the pods for the component are running. The following example lists the status of pods in the **openshift-user-workload-monitoring** project:

```
$ oc -n openshift-user-workload-monitoring get pods
```

NOTE

If an unrecognized **loglevel** value is included in the **ConfigMap** object, the pods for the component might not restart successfully.

2.9. NEXT STEPS

- Accessing monitoring for user-defined projects

CHAPTER 3. ACCESSING MONITORING FOR USER-DEFINED PROJECTS

By default, centralized monitoring for user-defined projects and platform monitoring are enabled. You can monitor your own projects in OpenShift Dedicated without the need for an additional monitoring solution.

The monitoring of user-defined projects cannot be disabled.

The **dedicated-admin** user has default permissions to configure and access monitoring for user-defined projects.

NOTE

Custom Prometheus instances and the Prometheus Operator installed through Operator Lifecycle Manager (OLM) can cause issues with user-defined project monitoring if it is enabled. Custom Prometheus instances are not supported.

3.1. NEXT STEPS

- Managing metrics

CHAPTER 4. MANAGING METRICS

This document provides an overview about how OpenShift Dedicated metrics are collected, queried and visualized.

4.1. UNDERSTANDING METRICS

In OpenShift Dedicated, cluster components are monitored by scraping metrics exposed through service endpoints. You can also configure metrics collection for user-defined projects. Metrics enable you to monitor how cluster components and your own workloads are performing.

You can define the metrics that you want to provide for your own workloads by using Prometheus client libraries at the application level.

In OpenShift Dedicated, metrics are exposed through an HTTP service endpoint under the **/metrics** canonical name. You can list all available metrics for a service by running a **curl** query against **http://<endpoint>/metrics**. For instance, you can expose a route to the **prometheus-example-app** example application and then run the following to view all of its available metrics:

```
$ curl http://<example_app_endpoint>/metrics
```

Example output

```
# HELP http_requests_total Count of all HTTP requests
# TYPE http_requests_total counter
http_requests_total{code="200",method="get"} 4
http_requests_total{code="404",method="get"} 2
# HELP version Version information about this binary
# TYPE version gauge
version{version="v0.1.0"} 1
```

Additional resources

- See the Prometheus documentation for details on Prometheus client libraries.

4.2. SETTING UP METRICS COLLECTION FOR USER-DEFINED PROJECTS

You can create a **ServiceMonitor** resource to scrape metrics from a service endpoint in a user-defined project. This assumes that your application uses a Prometheus client library to expose metrics to the **/metrics** canonical name.

This section describes how to deploy a sample service in a user-defined project and then create a **ServiceMonitor** resource that defines how that service should be monitored.

4.2.1. Deploying a sample service

To test monitoring of a service in a user-defined project, you can deploy a sample service.

Procedure

1. Create a YAML file for the service configuration. In this example, it is called **prometheus-example-app.yaml**.

2. Add the following deployment and service configuration details to the file:

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns1
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: prometheus-example-app
  name: prometheus-example-app
  namespace: ns1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: prometheus-example-app
  template:
    metadata:
      labels:
        app: prometheus-example-app
    spec:
      containers:
      - image: quay.io/brancz/prometheus-example-app:v0.2.0
        imagePullPolicy: IfNotPresent
        name: prometheus-example-app
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: prometheus-example-app
  name: prometheus-example-app
  namespace: ns1
spec:
  ports:
  - port: 8080
    protocol: TCP
    targetPort: 8080
    name: web
  selector:
    app: prometheus-example-app
  type: ClusterIP
```

   This configuration deploys a service named **prometheus-example-app** in the user-defined **ns1** project. This service exposes the custom **version** metric.

3. Apply the configuration to the cluster:

```
$ oc apply -f prometheus-example-app.yaml
```

It takes some time to deploy the service.

4. You can check that the pod is running:

```
$ oc -n ns1 get pod
```

Example output

```
NAME                                  READY   STATUS    RESTARTS   AGE
prometheus-example-app-7857545cb7-sbgwq   1/1     Running   0          81m
```

4.2.2. Specifying how a service is monitored

To use the metrics exposed by your service, you must configure OpenShift Dedicated monitoring to scrape metrics from the **/metrics** endpoint. You can do this using a **ServiceMonitor** custom resource definition (CRD) that specifies how a service should be monitored, or a **PodMonitor** CRD that specifies how a pod should be monitored. The former requires a **Service** object, while the latter does not, allowing Prometheus to directly scrape metrics from the metrics endpoint exposed by a pod.

NOTE

In OpenShift Dedicated, you can use the **tlsConfig** property for a **ServiceMonitor** resource to specify the TLS configuration to use when scraping metrics from an endpoint. The **tlsConfig** property is not yet available for **PodMonitor** resources. If you need to use a TLS configuration when scraping metrics, you must use the **ServiceMonitor** resource.

This procedure shows you how to create a **ServiceMonitor** resource for a service in a user-defined project.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role or the **monitoring-edit** role.
- For this example, you have deployed the **prometheus-example-app** sample service in the **ns1** project.

Procedure

1. Create a YAML file for the **ServiceMonitor** resource configuration. In this example, the file is called **example-app-service-monitor.yaml**.
2. Add the following **ServiceMonitor** resource configuration details:

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: prometheus-example-monitor
  name: prometheus-example-monitor
  namespace: ns1
spec:
  endpoints:
```

```
  - interval: 30s
    port: web
    scheme: http
  selector:
    matchLabels:
      app: prometheus-example-app
```

This defines a **ServiceMonitor** resource that scrapes the metrics exposed by the **prometheus-example-app** sample service, which includes the **version** metric.

3. Apply the configuration to the cluster:

```
$ oc apply -f example-app-service-monitor.yaml
```

It takes some time to deploy the **ServiceMonitor** resource.

4. You can check that the **ServiceMonitor** resource is running:

```
$ oc -n ns1 get servicemonitor
```

Example output

```
NAME                         AGE
prometheus-example-monitor   81m
```

Additional resources

- See the Prometheus Operator API documentation for more information on **ServiceMonitor** and **PodMonitor** resources.
- Accessing monitoring for user-defined projects.

4.3. QUERYING METRICS

The OpenShift monitoring dashboard lets you run Prometheus Query Language (PromQL) queries to examine metrics visualized on a plot. This functionality provides information about the state of a cluster and any user-defined projects that you are monitoring.

As a **dedicated-admin**, you can query one or more namespaces at a time for metrics about user-defined projects.

As a developer, you must specify a project name when querying metrics. You must have the required privileges to view metrics for the selected project.

4.3.1. Querying metrics for all projects as an administrator

As a **dedicated-admin** or as a user with view permissions for all projects, you can access metrics for all default OpenShift Dedicated and user-defined projects in the Metrics UI.

NOTE

Only dedicated administrators have access to the third-party UIs provided with OpenShift Dedicated Monitoring.

Prerequisites

- You have access to the cluster as a user with the **dedicated-admin** role or with view permissions for all projects.

Procedure

1. From the **Administrator** perspective in the OpenShift web console, select **Monitoring → Metrics**.
2. Select **Insert Metric at Cursor** to view a list of predefined queries.
3. To create a custom query, add your Prometheus Query Language (PromQL) query to the **Expression** field.
4. To add multiple queries, select **Add Query**.
5. To delete a query, select  next to the query, then choose **Delete query**.
6. To disable a query from being run, select  next to the query and choose **Disable query**.
7. Select **Run Queries** to run the queries that you have created. The metrics from the queries are visualized on the plot. If a query is invalid, the UI shows an error message.

NOTE

Queries that operate on large amounts of data might time out or overload the browser when drawing time series graphs. To avoid this, select **Hide graph** and calibrate your query using only the metrics table. Then, after finding a feasible query, enable the plot to draw the graphs.

8. Optional: The page URL now contains the queries you ran. To use this set of queries again in the future, save this URL.

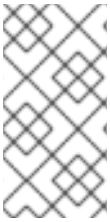
Additional resources

- See the Prometheus query documentation for more information about creating PromQL queries.

4.3.2. Querying metrics for user-defined projects as a developer

You can access metrics for a user-defined project as a developer or as a user with view permissions for the project.

In the **Developer** perspective, the Metrics UI includes some predefined CPU, memory, bandwidth, and network packet queries for the selected project. You can also run custom Prometheus Query Language (PromQL) queries for CPU, memory, bandwidth, network packet and application metrics for the project.



NOTE

Developers can only use the **Developer** perspective and not the **Administrator** perspective. As a developer you can only query metrics for one project at a time. Developers cannot access the third-party UIs provided with OpenShift Dedicated monitoring.

Prerequisites

- You have access to the cluster as a developer or as a user with view permissions for the project that you are viewing metrics for.
- You have enabled monitoring for user-defined projects.
- You have deployed a service in a user-defined project.
- You have created a **ServiceMonitor** custom resource definition (CRD) for the service to define how the service is monitored.

Procedure

1. From the **Developer** perspective in the OpenShift Dedicated web console, select **Monitoring → Metrics**.
2. Select the project that you want to view metrics for in the **Project:** list.
3. Choose a query from the **Select Query** list, or run a custom PromQL query by selecting **Show PromQL**.

NOTE

In the **Developer** perspective, you can only run one query at a time.

Additional resources

- See the Prometheus query documentation for more information about creating PromQL queries.
- See the Querying metrics for user-defined projects as a developer for details on accessing non-cluster metrics as a developer or a privileged user

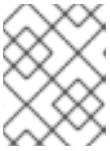
4.3.3. Exploring the visualized metrics

After running the queries, the metrics are displayed on an interactive plot. The X-axis in the plot represents time and the Y-axis represents metrics values. Each metric is shown as a colored line on the graph. You can manipulate the plot interactively and explore the metrics.

Procedure

In the **Administrator** perspective:

1. Initially, all metrics from all enabled queries are shown on the plot. You can select which metrics are shown.



NOTE

By default, the query table shows an expanded view that lists every metric and its current value. You can select ˅ to minimize the expanded view for a query.

- To hide all metrics from a query, click  for the query and click **Hide all series**.
- To hide a specific metric, go to the query table and click the colored square near the metric name.

2. To zoom into the plot and change the time range, do one of the following:
   - Visually select the time range by clicking and dragging on the plot horizontally.
   - Use the menu in the left upper corner to select the time range.
3. To reset the time range, select **Reset Zoom**.
4. To display outputs for all queries at a specific point in time, hover over the plot at that point. The query outputs appear in a pop-up box.
5. To hide the plot, select **Hide Graph**.

In the **Developer** perspective:

1. To zoom into the plot and change the time range, do one of the following:
   - Visually select the time range by clicking and dragging on the plot horizontally.
   - Use the menu in the left upper corner to select the time range.
2. To reset the time range, select **Reset Zoom**.
3. To display outputs for all queries at a specific point in time, hover over the plot at that point. The query outputs appear in a pop-up box.

Additional resources

- See the Querying metrics section on using the PromQL interface
- Troubleshooting monitoring issues

4.4. NEXT STEPS

- Alerts

CHAPTER 5. ALERTS

Alerts for monitoring workloads in user-defined projects are not currently supported in this product.

5.1. NEXT STEPS

- Reviewing monitoring dashboards

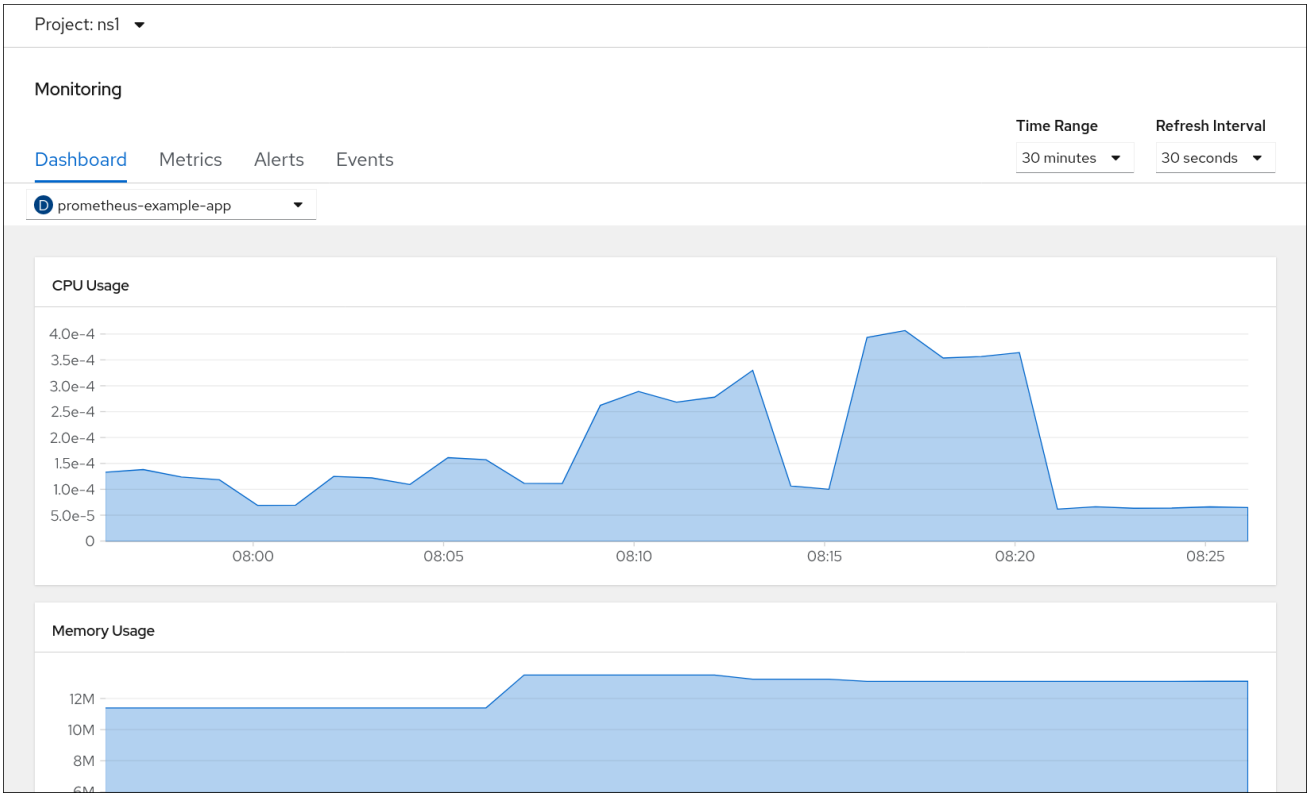
CHAPTER 6. REVIEWING MONITORING DASHBOARDS

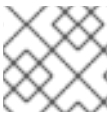
OpenShift Dedicated provides monitoring dashboards that help you understand the state of user-defined projects.

In the **Developer** perspective, you can access dashboards that provide the following statistics for a selected project:

- CPU usage
- Memory usage
- Bandwidth information
- Packet rate information

Figure 6.1. Example dashboard in the Developer perspective





NOTE

In the **Developer** perspective, you can view dashboards for only one project at a time.

6.1. REVIEWING MONITORING DASHBOARDS AS A DEVELOPER

In the **Developer** perspective, you can view dashboards relating to a selected project. You must have access to monitor a project to view dashboard information for it.

Prerequisites

- You have access to the cluster as a **dedicated-admin** or as a user with view permissions for the project that you are viewing the dashboard for.

Procedure

1. In the **Developer** perspective in the OpenShift Dedicated web console, navigate to **Monitoring → Dashboard**.
2. Choose a project in the **Project:** list.
3. Choose a workload in the **All Workloads** list.
4. Optional: Select a time range for the graphs in the **Time Range** list.
5. Optional: Select a **Refresh Interval**.
6. Hover over each of the graphs within a dashboard to display detailed information about specific items.

6.2. NEXT STEPS

- Troubleshooting monitoring issues

CHAPTER 7. TROUBLESHOOTING MONITORING ISSUES

This document describes how to troubleshoot common monitoring issues for user-defined projects.

7.1. DETERMINING WHY USER-DEFINED PROJECT METRICS ARE UNAVAILABLE

If metrics are not displaying when monitoring user-defined projects, follow these steps to troubleshoot the issue.

Procedure

1. Query the metric name and verify that the project is correct:

   a. From the **Developer** perspective in the OpenShift Container Platform web console, select **Monitoring → Metrics**.

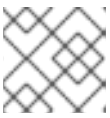
   b. Select the project that you want to view metrics for in the **Project:** list.

   c. Choose a query from the **Select Query** list, or run a custom PromQL query by selecting **Show PromQL**.
   The **Select Query** pane shows the metric names.

   Queries must be done on a per-project basis. The metrics that are shown relate to the project that you have selected.

2. Verify that the pod that you want metrics from is actively serving metrics. Run the following **oc exec** command into a pod to target the **podIP**, **port**, and /**metrics**.

```
$ oc exec <sample_pod> -n <sample_namespace> -- curl <target_pod_IP>:<port>/metrics
```



NOTE

You must run the command on a pod that has **curl** installed.

The following example output shows a result with a valid version metric.

Example output

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
# HELP version Version information about this binary-- --:--:-- --:--:--     0
# TYPE version gauge
version{version="v0.1.0"} 1
100   102  100   102    0     0  51000      0 --:--:-- --:--:-- --:--:-- 51000
```

An invalid output indicates that there is a problem with the corresponding application.

3. If you are using a **PodMonitor** CRD, verify that the **PodMonitor** CRD is configured to point to the correct pods using label matching. For more information, see the Prometheus Operator documentation.

4. If you are using a **ServiceMonitor** CRD, and if the **/metrics** endpoint of the pod is showing metric data, follow these steps to verify the configuration:

   a. Verify that the service is pointed to the correct **/metrics** endpoint. The service **labels** in this output must match the services monitor **labels** and the **/metrics** endpoint defined by the service in the subsequent steps.

   ```
   $ oc get service
   ```

   Example output

   ```
   apiVersion: v1
   kind: Service 1
   metadata:
     labels: 2
       app: prometheus-example-app
     name: prometheus-example-app
     namespace: ns1
   spec:
     ports:
     - port: 8080
       protocol: TCP
       targetPort: 8080
       name: web
     selector:
       app: prometheus-example-app
     type: ClusterIP
   ```

   1 Specifies that this is a service API.

   2 Specifies the labels that are being used for this service.

   b. Query the **serviceIP**, **port**, and **/metrics** endpoints to see if the same metrics from the **curl** command you ran on the pod previously:

      i. Run the following command to find the service IP:

      ```
      $ oc get service -n <target_namespace>
      ```

      ii. Query the **/metrics** endpoint:

      ```
      $ oc exec <sample_pod> -n <sample_namespace> -- curl <service_IP>:
      <port>/metrics
      ```

      Valid metrics are returned in the following example.

      Example output

      ```
      % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                     Dload  Upload   Total   Spent    Left  Speed
      100   102  100   102    0     0  51000      0 --:--:-- --:--:-- --:--:--   99k
      # HELP version Version information about this binary
      # TYPE version gauge
      version{version="v0.1.0"} 1
      ```

-

c. Use label matching to verify that the **ServiceMonitor** object is configured to point to the desired service. To do this, compare the **Service** object from the **oc get service** output to the **ServiceMonitor** object from the **oc get servicemonitor** output. The labels must match for the metrics to be displayed.
For example, from the previous steps, notice how the **Service** object has the **app: prometheus-example-app** label and the **ServiceMonitor** object has the same **app: prometheus-example-app** match label.

5. If everything looks valid and the metrics are still unavailable, please contact the support team for further help.